UCRL- 92007, REV. 1
PREPRINT

PARALLEL COMPUTATION OF MULTIPLE-SCALE PROBLEMS

R. C. Y. Chin
G. W. Hedstrom
J. R. McGraw
Lawrence Livermore National Laboratory

F. A. Howes
University of California Davis

This paper was prepared for submittal to:
The Proceedings of the ARO Workshop
on Parallel Computing
Stanford, CA
November 7-9, 1984

March 1, 1985

Parallel Computation of Multiple-Scale Problems

by

R. C. Y. Chin,[*] G. W. Hedstrom,[*]

F. A. Howes,[**] and J. R. McGraw[*]

Abstract. The efficient use of multiprocessor computers requires the introduction of new numerical algorithms, and the identification by asymptotic analysis of concurrencies inherent in the governing equations of multiple-scale problems is a good way to develop such algorithms. The language used must be able to maintain these concurrencies. Finally, depending on the problem and the computer, this concurrency may not be sufficient to keep all the processors busy, so that the language must be able to identify additional concurrencies. These ideas are illustrated in an example based on a convection-diffusion equation.

1. Introduction. Most major computer manufacturers now acknowledge that future generations of supercomputers will need to exploit parallel processing on a very large scale. We know from past experiences that such a drastic change in computers can produce major changes in the ways we design and write algorithms. For example, institutions which acquired a CDC Star computer had to redesign and rewrite essentially all of the programs to achieve promised performance levels. In the arena of multiprocessing, almost all of our experience is with vectorization — a very restricted form of parallelism. Here, we have knowledge about the impact of architectures on the selection of appropriate algorithms. Some work has been done to execute algorithms on general-purpose multiprocessors, but the emphasis there has been on showing how well the system can do with an existing algorithm. So far, the information we have is scattered and skewed.

There exist very many lines of FORTRAN code for scientific computation and much of this code will be difficult to adapt to multiprocessors. This is because the efficient use of multiprocessors requires the use of algorithms which minimize communication between processors and which have sufficient concurrency, i. e., the presence of independent subtasks which may be performed at the same time. We are faced with the choice of manually identifying concurrency and relying on extensions of FORTRAN to handle multiple tasks or of completely

[*]Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550

[**]Department of Mathematics, University of California Davis, Davis, CA 95616

rewriting the code in an applicative language. This decision clearly depends on the code or even on the section of the code. Thus, it is easy for a programmer or a compiler to find concurrency in a subroutine which does an explicit finite-difference calculation on a regular grid. For other numerical algorithms, however, it may be impossible for the compiler and very difficult for the programmer to find concurrency without major changes to the algorithms. In addition, we must point out that our limited experience indicates that it may also be extremely difficult to verify the reliability of codes using extensions of FORTRAN.

The immediate response to the arrival of multiprocessors is most likely to be an attempt to get the present FORTRAN codes to run on them with as little effort as possible. There will be a much bigger payoff, however, if physical processes which are nearly independent are identified and put on different processors with appropriate numerical methods. We should go back to the beginning and ask what is the information we want from the computation and what is the best way to obtain it.

From the point of view of computer science, the primary goals in the design and expression of algorithms for parallel processors are (1) minimization of the execution time, (2) minimization of global memory access, (3) minimization of communication between processors, and (4) identification of sufficient concurrency, i. e., identification of enough independent tasks to keep the processors busy. In short, we want to keep all of the processors working without forcing any to stop and wait for messages to be passed from memory or from other processors.

We must face the question of who is to partition the program into tasks and on what basis. It turns out that if we examine the original scientific problem with multiprocessing in mind, there are inherent partitions we can make, based on mathematical properties of the governing equations. The term multitasking is commonly used for breakup at this level. If the number of processors is sufficiently large, however, the degree of concurrency we get from multitasking will not be sufficient. Further concurrency will have to be supplied by the compiler. It should be kept in mind that in doing this further breakup the original multitasking identified by the analysis is to be maintained. Thus, we see that the efficient use of multiprocessors requires the consideration of issues in science, applied mathematics, and numerical analysis, as well as in computer science.

2. Multiple-Scale Problems. When we ask what problems of scientific computing are most suited to multitasking, we are led to examine multiple-scale problems because of the presence of nearly independent physical processes. Multiple-scale problems arise when several physical and/or chemical mechanisms are active, each associated with a particular scale. The relative importance of the various processes is measured by the ratio of the scales. When the ratio of the scales is a large or a small number, then one of the two competing mechanisms dominates. The local behavior is given by the dominant physical/chemical process. The essence of multiple-scale problems lies in the existence of distinguishable local competing mechanisms, acting on disparate temporal and spatial scales. The solutions of multiple-scale problems display different kinds of local behavior which may be rapidly varying depending on the relative magnitudes of the dimensionless numbers measuring the competing mechanisms. More often than not, the scales of the various competing processes and, therefore, the relative magnitudes of the dimensionless numbers change as the phenomenon evolves.

Operationally, we choose a set of scales commensurate with the physical/chemical mechanisms in question, and then cast the governing equations and the initial and boundary conditions in terms of dimensionless variables. If the resulting equations contain very large or very small dimensionless parameters, then the problem is a multiple-scale problem. These dimensionless parameters are ratios of scales, and they measure the importance of competing terms in the equations. Thus, for convection-diffusion equations the Peclet number or Reynolds number measures the relative importance of convection to diffusion. If, in addition, a chemical reaction is present, then the Damkohler number measures the relative importance of convection to reaction.

Current standard practice in solving partial differential equations involves the use of a single processor to do calculations with algorithms doing the same thing everywhere on a grid which has regular structure. This approach is quite adequate when the solution of the problem varies smoothly with respect to the grid everywhere in the domain, but it gets into difficulty when there are local regions of rapid spatial or temporal variation. Such local variations are a manifestation of the multiple-scale nature commonly found in the governing partial differential equations for problems of scientific computing. The computation of the solution using everywhere a grid and a method appropriate to the local regions of fastest variation becomes inefficient and at times impossible because of storage requirements. In fact, we may not even be interested in knowing the details of the rapid-scale motion, such as the structure of a shock, when only the speed and direction of the shock are of interest.

The issues involved in computing solutions of multiple-scale problems are examined in the survey paper by Chin et al. [3], and we give a brief summary here. It is possible to make a classification of multiple-scale problems from the point of view of the choices of numerical methods. We may also categorize multiple-scale problems for their parallel processing potential. This aspect is not considered in [3].

The most easily tractable multiple-scale problems are those in which there is only a small number of groups of scales which are widely separated and the motion on the fastest scales has little influence on the smooth part of the solution. An identifying feature of this class is the presence of regions in which the solution undergoes rapid variation. Such regions are called boundary or internal layers, depending on whether they are located near a boundary or in the interior of the domain. These are the problems which are most natural for multitasking because it makes sense to break up the domain according to the regions of different local behavior. An example of this type of problem is the propagation of a high-frequency wave in a medium with slowly varying wave speed. Here, the relevant length scale is the wavelength, which is short compared to the length $c/|\nabla c|$, which characterizes changes in the wave speed. The formation of caustics is an example of an internal layer in this setting.

A somewhat more difficult class to handle is that in which there is again a small number of groups of disparate scales, but the motion on the fast scales accumulates to induce a slow drift in the smooth part of the solution. Such problems exhibit rapidly varying solutions with a slowly varying envelope, and we want to compute only the envelope. In this case the most appropriate numerical technique involves some sort of averaging or homogenization of the differential equation. This may require occasional resolution of the rapid-scale behavior of the solution on local subdomains, as in Petzold [17] or Kirchgraber [12]. It is not clear how best to implement the solution of such problems on a multiprocessor. An example of this kind of problem is the propagation of a low-frequency wave in a medium with rapidly varying wave speed. The rapid local variations have little effect over a short time, but the long-time influence of the scattering of the wave is substantial. In this case we are not interested in the details of the local motion, but we want only to compute the general drift of the solution.

Finally, the most difficult multiple-scale problems are those in which there is a broad range of scales, rather uniformly distributed. Turbulence, for example, falls into this category, as does wave propagation in a varying medium when the frequency content of the wave extends over a broad range. Little can be said in general about methods for solving such problems, but in particular cases, such as the ray-mode representation of the solution of wave-propagation problems by Felsen and Kamel [7], it is useful to make projections onto subdomains of the solution space.

For special multiple-scale problems there are standard tricks, such as the use of a grid which resolves the rapid variation. For example, in the computation of steady flight the boundary layer near an airfoil is commonly handled by the use of body-fitted coordinates with a fine grid in the direction normal to the airfoil. This is effective because it is known from asymptotic analysis of the governing equations that the main flow is not influenced appreciably by the behavior in the boundary layer, but the effects of viscosity must be represented accurately if we are interested in separation phenomena. In addition, the viscous effects act primarily in the direction normal to the airfoil. We need to extend these ideas to more general situations, and doing so requires asymptotic analysis of the governing equations to determine how best to break up a problem into smaller problems on subdomains and to give guidance in the selection of numerical algorithms on these subdomains. It may even turn out to be appropriate to use an implicit method on one subdomain and explicit methods on its neighbors. This breaking up of a problem into components is appropriate even for a serial computer, but it is a natural thing to do on a multiprocessor.

The multiple-scale problems with a number of groups of scales which are widely separated belong to the class of weakly interacting systems. For problems in which the motion on the fastest scales has little influence on the smooth part of the solution, such as internal and boundary layers, the domain is naturally divided into subregions of different behavior which are weakly coupled to each other. This automatically gives some degree of concurrency, and the more complicated the behavior of the solution —the more concurrency. We can get a rough idea of the concurrency available here by making work estimates, but we have to implement the algorithms on a machine in order to find out the precise dividing line.

For the multiple-scale problems involving slowly modulated rapid variation it may be feasible to do computations which resolve the rapid motion on representative subdomains, with different processors working on different subdomains. This method would be an extension to partial differential equations of the ideas of [17] and [12] for ordinary differential equations. This is an area which is completely open at this time

In the case of multiple-scale problems with a wide range of scales, rather uniformly distributed, it may be possible to use a ray-mode type of subspace projection. This would be an extension of the work of Felsen and Kamel [7] for wave-propagation problems. Such an algorithm would be based on a decomposition in which some processors do a modal analysis on the slow variation while others do a ray expansion on the fast variation. Certainly, the ray-mode [7] expansion for linear wave-propagation problems may be implemented on multiprocessors. For nonlinear problems of this kind, this algorithm may again be used but as part of an iterative method.

In any case, the essential mathematical technique for solving multiple-scale problems is the subspace projection method. Corresponding to a distinguishable local mechanism there is an associated subspace in the space of solutions. The determination of the structure of the subspace is governed by analysis, both mathematical and numerical. It may be convenient to view the splitting into subspaces differently in different settings for the purpose of developing an accurate and efficient algorithm. In particular for linear problems, the different subspaces may arise from a decomposition of the domain of definition or from a spectral decomposition.

In domain decomposition, we solve locally approximate equations whose solutions generate a subspace which may be identified as an approximation to a subspace in the spectral decomposition of the original problem. The use of domain decomposition gives an additional degree of freedom in the numerical solution, namely, we may solve the local equations using whatever method to gain accuracy and efficiency. The spectral decomposition method permits a convenient way to analyze approximation errors and to view concurrency.

In applying subspace projection by spectral methods we are led to the computation of a basis for each subspace, projection operators onto the subspace, and generalized Fourier coefficients. Of course, the selection of the local basis and its computation are based on asymptotic analysis. The calculations of the projections and the generalized Fourier coefficients involve inner products, which typically use fan-in algorithms. That is, the concurrency is high to begin with, but it diminishes as the computation proceeds. This implies that some of the concurrency must be supplied by the language or by the compiler.

Note that it is not necessary to regard domain decomposition from a spectral point of view, and in our example in Section 6 we take a direct approach.

3. Partitioning: Options and Limits. In this section we examine in greater detail the question of identification of concurrency by the programmer and by the language. Intelligent compiler research efforts all use essentially the same key principle: only data dependencies need limit the degree to which operations can proceed simultaneously. If operation B needs the result of operation A, then B cannot execute simultaneously with A. At the highest level, this principle controls Kuck's efforts [13], just as it does applicative language work (McGraw [14]), even though both use it in very different ways. Implicit in this general approach to partitioning is one important fact. Intelligent compilers must work with the specific algorithm described by a program — compilers cannot find concurrency in an inherently sequential program. Thus, programmers must still write programs with attention to issues of concurrency. Another point of concern is that most of this kind of research uses existing algorithms for benchmarking progress (because they are available). It is not at all clear that these task-division algorithms will work as well on new algorithms designed with parallel computation in mind.

In contrast, however, programmer-based partitioning is still in its infancy. Most of the language tools given to programmers came out of research in operating systems — a very different problem domain. These tools allow programmers to describe concurrency and synchronization that make the partitioning obvious. But with the slightest misuse, these tools produce unreproducible results, deadlocks, and other equally frustrating occurrences. Almost no language design work has been based on considerations of exactly what types of concurrency these new applications programs are likely to need.

4. Concurrency and Synchronization: Language Options. Probably the greatest source of background information relevant to scientific computing with multiprocessors is in the area of language options for parallel processing. In an excellent survey paper Andrews and Schneider [1] discuss and evaluate most of the significant options. The apportionment of the content of this paper clearly indicates where the difficult problems lie; only a minor portion of the article discusses options for creating concurrent tasks, while a major portion discusses options for synchronization. The latter is a much more difficult task to do correctly. We can roughly divide the strategies for handling synchronization into three groups: shared-memory options, message-passing options, and data-flow options. These options lead to very different types of tradeoffs in the following areas: ways of expressing concurrency, memory requirements, amounts of concurrency, and the consequences of misuse.

The shared-memory model assumes that all processors access a large amount of global memory. The synchronization primitives allow programmers to coordinate accesses to that memory so that some timing errors can be avoided. Examples of such primitives include: semaphores, regions, and monitors. These schemes vary in the degree to which they ensure that a program accesses the shared memory in safe, repeatable patterns. In particular, semaphores are often proposed as the simplest scheme to provide to users. Regions and monitors afford more protection, because they prevent two processors from getting simultaneous access to shared data. The trouble with the shared-memory model is that none of these three tools can prevent the user from writing code for which the final answer depends on the order in which the individual subtasks happen to be performed. The shared-memory model does have the advantage that it encourages users to think in terms of each processor having access to the entire problem being solved. During execution each processor can usually switch its focus to any area of the calculation quite easily. Therefore, more of the effort of partitioning may be done during program execution.

In contrast, the message-passing model generally assumes no global shared memory. This view corresponds to the distributed computing world where several large computers are connected by some form of high-speed channel. In this model, processors communicate by passing messages to each other, and the global data for the system must be divided among the various processors. Programs that use message passing are still susceptible to timing problems that can lead to unrepeatable behavior, but under this strategy the causes are often easier to identify than with shared memory. Partitioning in this system is often more difficult because the programmer needs to identify divisions that decrease the amount of message traffic while keeping work loads relatively even.

The data-flow model approaches synchronization in a totally different manner. Both concurrency and synchronization are implicit in a program, rather than explicit. The rules governing each are the same. All operations in the language are treated as functions that map inputs to outputs. Operations whose inputs do not depend on each other's outputs (either directly or indirectly) can execute simultaneously. If the output for a function is an input to another function, then the second function will be synchronized to execute after the first has finished. This type of concurrency ensures determinate program behavior, but unless extended in some fashion, it also precludes indeterminate algorithms like chaotic relaxation techniques. To date there has been insufficient testing to determine how this model influences the programming style of users.

We need to ask how the forms of synchronization and communication commonly available in programming languages alter the mathematical and numerical formulation of the problems. Synchronization and communication between parallel processors can be the rate-limiting steps to effective computation. Synchronization is required for coordinating the activities and for timing data transfer between related processors. A process can't be continued or started if the required data is delayed. This gives rise to work stoppage and, therefore, a reduction in available processor utilization.

For effective decomposition of computational tasks, an asynchronous algorithm with local access of information is preferred. Translating these into mathematical requirements, we observe that problems governed by hyperbolic partial differential equations with their local domains of influence are most likely to achieve full parallel processing capability. Elliptic problems, on the other hand, with their global structure are the least likely to be suitable for parallel processing in the sense of asynchronization and local communication.

In numerical analysis, synchronization and communication affect the design of an algorithm. In the case of matrix multiplication on the multiprocessor Cm*, Ostlund, Hibbard and Whiteside [16] have found that communication and synchronization are, indeed, the rate-limiting steps in an $O(\log_2 N)$ algorithm. On the other hand, the analysis of this algorithm by Sameh [18] and Heller [9] focused only on arithmetic operation counts while neglecting communication and synchronization costs. Two alternate schemes for taking into account the communication and synchronization costs are proposed in [16]. One of these methods is implemented on the Cm* multiprocessor. Both methods partition the matrices so as to reduce nonlocal memory access.

Another example of an algorithm formulation altered by synchronization and communication among related processors is the solution by finite differences of Laplace's equation on a plane domain with Dirichlet boundary conditions. Here, an asynchronous iterative scheme is developed by Baudet [2]. Experimental results have shown that this asynchronous iterative scheme performs far better than synchronous iterations. Unfortunately, proof of convergence for asynchronous iterative schemes is more difficult than for synchronous iterations.

Other avenues of attack are available for solving the elliptic problem. In particular, because of its nature the subspace projection method may well be competitive with the asynchronous methods in terms of accuracy and efficiency. Here, we can control the accuracy by examining the rate of decay of the generalized Fourier coefficients of the boundary data. This determines the number of terms needed in the spectral representation of the solution to attain the desired accuracy. Moreover, the differential equations for the coefficients of the eigenfunction expansion are totally independent of each other, so that only access to local memory is needed. This gives an asynchronous algorithm with local memory access, in accordance with the requirements for effective multiprocessing.

5. A Singularly Perturbed Parabolic Equation — Theoretical Background. As an example to illustrate how asymptotic analysis may be used to identify concurrency and suggest numerical methods, we examine a convection-diffusion equation. In terms of dimensional variables consider the partial differential equation

$$\hat{u}_{\hat{t}} + \hat{c}(\hat{x},\hat{t})\hat{u}_{\hat{x}} = \hat{\varepsilon}\hat{u}_{\hat{x}\hat{x}} \tag{5.1}$$

on the domain

$$\hat{D} = \{(\hat{x},\hat{t}) \mid 0 < \hat{x} < \hat{b}, \quad 0 < \hat{t} < \hat{T}\},$$

with initial and boundary conditions

$$\begin{aligned} \hat{u}(0,\hat{t}) &= \hat{\alpha}(\hat{t}), \quad 0 < \hat{t} < \hat{T}, \\ \hat{u}(\hat{b},\hat{t}) &= \hat{\beta}(\hat{t}), \quad 0 < \hat{t} < \hat{T}, \\ \hat{u}(\hat{x},0) &= \hat{\gamma}(\hat{x}), \quad 0 < \hat{x} < \hat{b}. \end{aligned} \tag{5.2}$$

For convenience, the functions $\hat{\alpha}$, $\hat{\beta}$, $\hat{\gamma}$, and $\hat{c}$ are assumed to be C^∞ functions. It is also assumed that the boundary data is compatible with the initial data in the sense that

$$\hat{\alpha}(0) = \hat{\gamma}(0), \quad \hat{\beta}(0) = \hat{\gamma}(\hat{b}).$$

Our analysis is closely related to the ideas of matched asymptotic expansions (see Eckhaus [6], Kevorkian and Cole [11], and Nayfeh [15]), but we do the matching numerically. We introduce dimensionless variables and identify the parameters which measure ratios of scales. Thus, $\hat{\varepsilon}$ is a positive number which is small in a sense which will be made precise later. In this section we discuss asymptotic analysis of the case when the convection dominates the diffusion, and in the next section we show how to use this information to design a numerical method exploiting concurrency.

In preparation for the scaling let us do a qualitative analysis of the problem. This analysis leads to an initial division of $\hat{D}$ into the union of two subdomains. Since we are primarily interested in the case in which convection dominates over diffusion, we begin with an examination of the reduced equation

$$\hat{u}_{\hat{t}} + \hat{c}(\hat{x},\hat{t})\hat{u}_{\hat{x}} = 0 \tag{5.3}$$

on $\hat{D}$. The solutions $\hat{U}$ of (5.3) are constant on the characteristic curves [5], i. e., on the trajectories of the ordinary differential equation

$$\frac{d\hat{x}}{d\hat{t}} = \hat{c}(\hat{x},\hat{t}) \tag{5.4}$$

Thus, the direction of propagation depends on the sign of $\hat{c}$. For convenience we assume that $\hat{c}>0$ on $\hat{D}$, so that the convection is in the direction of increasing $\hat{x}$. Then, for (5.3) we may impose the initial and boundary conditions,

$$\begin{aligned} \hat{u}(\hat{x},0) &= \hat{\gamma}(\hat{x}), \quad 0 < \hat{x} < \hat{b}, \\ \hat{u}(0,\hat{t}) &= \hat{\alpha}(\hat{t}), \quad 0 < \hat{t} < \hat{T}, \end{aligned} \tag{5.5}$$

but the boundary condition at the outflow,

$$\hat{u}(\hat{b},\hat{t}) = \hat{\beta}(\hat{t}), \quad 0 < \hat{t} < \hat{T}$$

may not be satisfied. These considerations suggest a partition of $\hat{D}$ by the characteristic $\hat{x}=\Gamma(\hat{t})$ which is the solution to (5.4) with $\hat{x}(0)=0$. Note that the characteristic curve $\hat{x}=\hat{\Gamma}(\hat{t})$ exits $\hat{D}$ either through the side $\hat{x}=\hat{b}$ or through the top $\hat{t}=\hat{T}$. Let us restrict our attention to the case when the the exit is through $\hat{x}=\hat{b}$, which turns out to be the slightly more complicated case of the two. We therefore define two subdomains, according to whether the solution $\hat{U}$ of (5.3) is influenced by $\hat{\alpha}$ or by $\hat{\gamma}$,

$$\Omega_\gamma = \{(\hat{x},\hat{t}) \text{ in } \hat{D} \quad 0 < t < \hat{\Gamma}^{-1}(\hat{x})\}$$

and

$$\Omega_\alpha = \{(\hat{x},\hat{t}) \text{ in } \hat{D} \mid \hat{\Gamma}^{-1}(\hat{x}) < \hat{t} < \hat{T}\}.$$

These domains are illustrated in Fig. 1.

In preparation for a theoretical discussion of the difference between the solution to (5.1-2) and the solution to (5.3-4), we perform an initial scaling in $\hat{D}$. Eq. (5.1) is invariant under the transformation

$$\hat{u} = Wu + a,$$

and this transformation induces analogous transformations of the initial and boundary data,

$$\hat{\alpha} = W\alpha + a,$$

$$\hat{\beta} = W\beta + a,$$

$$\hat{\gamma} = W\gamma + a.$$

At this point we have to decide what we wish to emphasize. Thus, assuming that we are interested in the evolution of the initial data $\hat{\gamma}$ and that $\hat{\gamma}$ is not constant, we may choose W and a so that

$$0 \leq \gamma(\hat{x}) \leq 1, \quad 0 < \hat{x} < \hat{b},$$

where the upper and lower bounds are each attained for at least one value of $\hat{x}$ in $[0,\hat{b}]$. Assume that with this normalization we have $|\alpha(\hat{t})| \leq 1$ for $0 \leq \hat{t} \leq \hat{T}$.

Let us now examine the spatial and temporal scales. In Ω_γ there are several length scales, including the length $\hat{b}$ of the interval and $L_\gamma = \min 1/|\gamma'(\hat{x})|$ associated with variations in the initial data γ. Similarly, in Ω_α there are obvious time scales, the time duration $\hat{T}$ and $T_\alpha = \min 1/|\alpha'(\hat{t})|$ associated with the boundary data α. In $\hat{D}$ there are also several, not so obvious, length scales associated with the variations of the convection velocity $\hat{c}$ and its derivative $\partial\hat{c}/\partial\hat{x}$. We write $\hat{c} = Vc$ with V chosen so that

$$V = \sup_{\hat{D}} \hat{c}.$$

The length scale measuring the variation of c is given by

$$L_c = \inf_{\hat{D}} \frac{1}{|\partial c/\partial\hat{x}|}.$$

We shall see that the length scale

$$L_{dc} = \frac{\sup_{\hat{D}} |\partial c/\partial\hat{x}|}{\sup_{\hat{D}} |\partial^2 c/\partial\hat{x}^2|}$$

must also be considered. Their roles in the solution will become clear in the next section.

Corresponding to the relevant time or length scale in Ω_α and Ω_γ, there are length or time scales induced by convection and diffusion. In Ω_γ, there is a convective time scale $T_\gamma = L_\gamma / V$ and a diffusive time scale $T_{\gamma\varepsilon} = L_\gamma^2 / \hat{\varepsilon}$. Similarly, in Ω_α there is a convective length scale $L_\alpha = VT_\alpha$ and a diffusive length scale $L_{\alpha\varepsilon} = \sqrt{\hat{\varepsilon}T_\alpha}$.

Because we are primarily interested in the evolution of the initial data $\hat{\gamma}$, we change variables in $\hat{D}$ according to the scales induced by $\hat{\gamma}$ and $\hat{c}$. Thus, we set

$$\hat{x} = L_\gamma x \quad \text{and} \quad \hat{t} = T_\gamma t,$$

mapping $\hat{D}$ onto a rectangle

$$D = \{(x,t) \mid 0 < x < b,\ 0 < t < T\}.$$

On D (5.1) is transformed into an equation of the form

$$\frac{\partial u}{\partial t} + c(x,t)\frac{\partial u}{\partial x} = \varepsilon \frac{\partial^2 u}{\partial x^2}, \tag{5.6}$$

with $\varepsilon = \hat{\varepsilon} / VL_\gamma$. The initial and boundary conditions for (5.6) in D are

$$\begin{aligned} u(x,0) &= \gamma(x), \quad 0 < x < b, \\ u(0,t) &= \alpha(t), \quad 0 < t < T, \quad \text{and} \\ u(b,t) &= \beta(t), \quad 0 < t < T. \end{aligned} \tag{5.7}$$

The other scales now appear in the form of dimensionless parameters — ratios with respect to the fundamental scales. Thus, b is the ratio of the length $\hat{b}$ of the original interval to the characteristic length L_γ of the initial data $\hat{\gamma}$. Because we are primarily interested in the evolution of the initial data $\hat{\gamma}$, we assume that $b \gg 1$. The other dimensionless parameter in (5.6), ε, is the reciprocal of the Reynolds number. It is the ratio of the induced diffusion time $T_{\gamma\varepsilon}$ to the induced convection time T_γ, and it measures the importance of diffusion relative to convection. Note that if $0 < \varepsilon \ll 1$ (or $T_\gamma \ll T_{\gamma\varepsilon}$), then (5.6) is a singularly perturbed parabolic equation in D, and the reduced equation for (5.6) obtained by setting $\varepsilon = 0$ is

$$\frac{\partial u}{\partial t} + c(x,t)\frac{\partial u}{\partial x} = 0. \tag{5.8}$$

The initial and boundary conditions for (5.8) are

$$\begin{aligned} u(x,0) &= \gamma(x), \quad 0 < x < b, \\ u(0,t) &= \alpha(t) \quad 0 < t < T. \end{aligned} \tag{5.9}$$

For the sake of simplicity we impose the condition that $c(x,t) \geq c_0 > 0$ in a strip

$$B = \{(x,t) \mid x_1 \leq x \leq b,\ 0 < t < T\}$$

including the outflow boundary of D. This condition forces the characteristics to exit D transversally. The transversality of the characteristics and the need to satisfy the remaining boundary condition $u(b,t) = \beta(t)$ give rise to an "exponential" boundary layer for (x,t) in B. This boundary layer may be seen in terms of a rescaling of (5.6) in terms of $x = b - \varepsilon\eta$.

$$\varepsilon \frac{\partial u}{\partial t} - c(b - \varepsilon\eta, t)\frac{\partial u}{\partial \eta} = \frac{\partial^2 u}{\partial \eta^2}.$$

Therefore, we are led to expect that in B the solution to (5.6-7) may be approximated for ε sufficiently small by the solution to the reduced equation

(5.10) $$-c(b,t)\frac{\partial u}{\partial \eta} = \frac{\partial^2 u}{\partial \eta^2}$$

with boundary data

$$u = \beta(t) \quad \text{at } \eta = 0$$

and u at $\eta = (b - x_1)/\varepsilon$ as obtained from the solution to (5.8-9). There will, of course, be an initial layer near $t=0$ in which (5.10) is not a valid approximation to (5.6), because we have lost the initial condition. All of this heuristic analysis is justified by the following theorem of Howes [10].

THEOREM 5.1. Suppose that the problem (5.8-9) has a smooth solution $U(x,t)$ in D. Then (5.6-7) has a smooth solution $u(x,t,\varepsilon)$, and there exists a positive constant C such that

(5.11) $$|u(x,t,\varepsilon) - U(x,t)| \le C\varepsilon$$

for (x,t) in $D \setminus B$, and

(5.12) $$|u(x,t,\varepsilon) - U(x,t)| \le K(t)\exp\{-k(b-x)/\varepsilon\} + C\varepsilon$$

for (x,t) in B with $K(t) = |U(b,t) - \beta(t)|$ and $k < c_0$.

Let us add another complication — an internal "corner" layer generated by a discontinuity in ∇U at the origin. Such a discontinuity may easily arise, because (5.8) prescribes the directional derivative of U along the characteristic curves, while the initial and boundary data (5.9) specify the directional derivatives of U along the coordinate axes. Thus, it is easy to see that ∇U is continuous at the origin if and only if $\alpha(0) = \gamma(0)$ and

$$\frac{d\alpha}{dt} = -c\frac{d\gamma}{dx}$$

at the origin. In the case when the solution U of (5.8-9) is continuous but ∇U is discontinuous along $x = \Gamma(t)$, in place of (5.11) and (5.12) we have [10]

(5.13) $$|u(x,t,\varepsilon) - U(x,t)| \le$$

$$C\varepsilon + K(t)\exp\{-k(b-x)/\varepsilon\} + C_1\varepsilon^{1/2}\exp\{-\frac{|x - \Gamma(t)|}{\varepsilon^{1/2}}\}$$

for (x,t) in D.

6. A Singularly Perturbed Parabolic Equation — Domain Decomposition. In this section the theoretical tools of the previous section are used to develop a numerical method for the problem (5.1-2) which contains a high degree of concurrency. We show that there may be different spatial and temporal scales in different subdomains and that domain decomposition may be based on this fact. In addition, we show that in the different subdomains we want to use different grid sizes in both the spatial and temporal directions, and we may even want to use different numerical methods.

In applying the domain decomposition method, the domain of definition is partitioned into subdomains. Along interior subdomain boundaries, the solution and its derivative $\partial\hat{u}/\partial\hat{x}$ are continuous. This is the patching of subdomain solutions to form a global solution. For singular perturbation problems, it is often not necessary to ensure that the spatial derivative is, indeed, continuous. This is because the continuity of $\partial\hat{u}/\partial\hat{x}$ is asymptotically satisfied. We rely heavily on a priori estimates to guide us in dealing with the domain partitioning strategy.

The estimate (5.13) indicates that the domain $\hat{D}$ should be subdivided as shown in Fig. 2. More specifically, we select numbers $\hat{x}_0$, $\hat{x}_1$, and $\hat{t}_0$ such that $0<\hat{x}_0<\hat{x}_1<\hat{b}$ and $0<\hat{t}_0<\hat{T}$. Let $\hat{x}=\hat{\Gamma}_\gamma(\hat{t})$ be the characteristic curve for (5.3) through $(\hat{x}_0,0)$, i. e., the solution of (5.4) with initial data $\hat{x}(0)=0$. Similarly, let $\hat{x}=\hat{\Gamma}_\alpha(\hat{t})$ be the characteristic curve for (5.3) through the point $(0,\hat{t}_0)$. Then we define the following subdomains,

$$\hat{D}_\gamma = \{(\hat{x},\hat{t}) \mid \hat{\Gamma}_\gamma(\hat{t}) < \hat{x} < \hat{x}_1,\ 0 < \hat{t} < \hat{\Gamma}_\gamma^{-1}(\hat{x}_1)\},$$

$$\hat{D}_\alpha = \{(\hat{x},\hat{t}) \mid 0 < \hat{x} < \hat{x}_1,\ \hat{\Gamma}_\alpha^{-1}(\hat{x}) < \hat{t} < \hat{T}\},$$

$$\hat{D}_{\alpha\gamma} = \{(\hat{x},\hat{t}) \mid \max[0,\hat{\Gamma}_\alpha(\hat{t})] < \hat{x} < \min[\hat{x}_1,\ \hat{\Gamma}_\gamma(\hat{t})],\ 0 < \hat{t} < \hat{\Gamma}_\alpha^{-1}(\hat{x}_1)\},$$

$$\hat{D}_{\beta\gamma} = \{(\hat{x},\hat{t}) \mid \hat{x}_1 < \hat{x} < \hat{b},\ 0 < \hat{t} < \hat{\Gamma}_\gamma^{-1}(\hat{x}_1)\},$$

$$\hat{D}_{\alpha\beta} = \{(\hat{x},\hat{t}) \mid \hat{x}_1 < \hat{x} < \hat{b},\ \hat{\Gamma}_\alpha^{-1}(\hat{b}) < \hat{t} < \hat{T}\},$$

$$\hat{D}_{\alpha\beta\gamma} = \{(\hat{x},\hat{t}) \mid \hat{x}_1 < \hat{x} < \hat{b},\ \hat{\Gamma}_\gamma^{-1}(\hat{x}_1) < \hat{t} < \hat{\Gamma}_\alpha^{-1}(\hat{b})\}.$$

The selection of $\hat{x}_0$, $\hat{x}_1$, and $\hat{t}_0$ is as follows. Based on (5.13) and a given error tolerance, we choose the boundaries of $\hat{D}_\alpha$ and $\hat{D}_\gamma$ so that the solution $\hat{U}$ to (5.3-5) is a sufficiently accurate approximation to the solution $\hat{u}$ to (5.1-2) in the set $\hat{D}_\alpha \cup \hat{D}_\gamma$. It should be noted, however, that the scalings in Section 5, leading up to the definition of ε, were based on global properties of $\hat{\alpha}$, $\hat{\gamma}$, and $\hat{c}$. Thus, since we are splitting $\hat{D}$ into subdomains anyway, we ought to use independent scalings in the different subdomains. Consequently, we could obtain sharper versions of (5.13), valid separately in the domains $\hat{D}_\gamma$, $\hat{D}_{\beta\gamma}$, $\hat{D}_\alpha$, $\hat{D}_{\alpha\beta}$, $\hat{D}_{\alpha\gamma}$, and $\hat{D}_{\alpha\beta\gamma}$. In this way, we might well be able to reduce the thickness of the subdomains encompassing the boundary and internal layers. Note also that the use of independent scaling in nearly independent subdomains gives us the freedom to use grids of different mesh sizes in different subdomains.

Based on Theorem 5.1, we construct a very simple numerical method which is valid under the conditions of the theorem when ε is sufficiently small. In $\hat{D}_\alpha$ and $\hat{D}_\gamma$ we solve the reduced equation (5.3) with boundary and initial data (5.5). The solutions to these two problems provide the boundary data needed to solve (5.1) in the domains $\hat{D}_{\beta\gamma}$, $\hat{D}_{\alpha\gamma}$, and $\hat{D}_{\alpha\beta}$. We shall say more about the method to be used in $\hat{D}_{\alpha\gamma}$ in a moment; for now it suffices to say that the solution in $\hat{D}_{\alpha\gamma} \cup \hat{D}_\alpha$ provides the boundary data for the problem in $\hat{D}_{\alpha\beta\gamma}$. In $\hat{D}_{\beta\gamma}$ we have to solve the full equation (5.1) in an initial layer near the $\hat{x}$-axis, but outside this layer we use the solution to (5.10) with the required boundary conditions. In $\hat{D}_{\alpha\beta\gamma}$ we simply solve (5.1), providing the initial conditions for the problem on $\hat{D}_{\alpha\beta}$. Finally, in $\hat{D}_{\alpha\beta}$ we use the same method as was used on $\hat{D}_{\beta\gamma}$.

Let us consider the problem on $\hat{D}_{\alpha\gamma}$ in more detail. In order to avoid needless complication of the notation, our discussion is based on the scaled equation (5.6) on the image $D_{\alpha\gamma}$ of $\hat{D}_{\alpha\gamma}$, even though, as stated earlier, we might well want to use different scalings in different subdomains. Let us introduce a coordinate system based on the characteristics for (5.6). Thus, we define a mapping from the (x,t)-domain $D_{\alpha\gamma}$ to a (ξ,τ)-domain $\Delta_{\alpha\gamma}$. For $\xi \geq 0$ we use the characteristic curves which intercept the x-axis, i. e., we take $t=\tau$ and

$$\frac{\partial x}{\partial \tau} = c(x(\xi,\tau),\tau) \tag{6.1}$$

with $x(0)=\xi$. We make a similar transformation for $\xi<0$, but before we do so, we must specify the image of the boundary $x=0$, $0 \leq t \leq T$. One convenient way to do this is based on the fact that along the line $\tau=0$ for $\xi \geq 0$ the differential of the mapping defined by (6.1) is

$$dt = d\tau, \quad dx = d\xi + c(x,0)d\tau$$

In order to ensure that the mapping is smooth in a neighborhood of the origin, we require that on the line $x=0$ the differential of the mapping is

$$dt = d\tau, \quad dx = d\xi + c(0,t)d\tau$$

That is,the image of the line $x=0$ is taken to be the solution curve of the ordinary differential equation

$$\frac{d\xi}{d\tau} = -c(0,\tau), \quad \xi(0) = 0. \tag{6.2}$$

Let us denote the solution to (6.2) by $\xi = y_0(\tau)$. Because $c>0$, it follows that y_0 is a monotonically decreasing function and that $y_0(\tau)<0$ for $\tau>0$. In order to ensure that the mapping defined by (6.1) will be C^1 at the origin, as initial data for (6.1) for $\xi<0$ we take $x=0$ at $\tau=y^{-1}(\xi)$. Let us remark that from standard theorems on the smooth dependence of solutions of ordinary differential equations on initial data [4], it follows that the mapping we have constructed is C^1 in $D_{\alpha\gamma}$.

The Jacobian of our transformation is $J=\partial x/\partial\xi$, and our construction of initial conditions for (6.1) implies that $J=1$ on the lines $t=0$ and $x=0$. With this transformation Eq. (5.6) becomes

$$\frac{\partial u}{\partial \tau} = \frac{\varepsilon}{J}\frac{\partial}{\partial \xi}\left[\frac{1}{J}\frac{\partial u}{\partial \xi}\right], \tag{6.3}$$

or equivalently,

$$\frac{\partial u}{\partial \tau} = \frac{\varepsilon}{J^2}\left[\frac{\partial^2 u}{\partial \xi^2} - \frac{1}{J}\frac{\partial J}{\partial \xi}\frac{\partial u}{\partial \xi}\right] \tag{6.4}$$

It should be noted that this transformation to characteristic coordinates may be done on any of our six subdomains and that the form of (6.3) remains the same. (There is, however, no advantage in making this transformation in B.) Only ε changes as we use different scalings on different subdomains. In fact, on $D_\alpha \cup D_\gamma$ we propose solving only the reduced form of (6.3),

$$\frac{\partial u}{\partial \tau} = 0. \tag{6.5}$$

In this form of the equation we see a very high degree of concurrency.

The image of $D_{\alpha\gamma}$ in the characteristic coordinate system is a domain

$$\Delta_{\alpha\gamma} = \{(\xi,\tau) \mid y_0(t_0) < \xi < x_0, \ \max[0, y_0^{-1}(\xi)\,] < \tau < y_1^{-1}(\xi)\},$$

where $\xi=y_1(\tau)$ is the image of the boundary $x=x_1$. For $\xi \geq 0$ the initial data for (6.3) is $u(\xi,0)=\gamma(\xi)$, and for $\xi<0$ it is $u(y_0(\tau),\tau)=g(\tau)$. The boundary data is provided by the solutions to (6.5) in the images of $\hat{D}_\alpha$ and $\hat{D}_\gamma$. Regarding practical numerical algorithms for (6.3) in $\Delta_{\alpha\gamma}$, let us remark that we expect $\partial u / \partial \tau$ to be small except in a region of layer development near the origin. Consequently, we would use a small time step only in this neighborhood, and we would use the hopscotch method of Gourlay [8] outside this neighborhood. This is because the hopscotch method is an explicit method which is stable for arbitrary time steps, but it is only first-order accurate with respect to the timestep.

Note that our analysis also forms the basis of a more robust method, which may be used when the condition of Theorem 5.1 (smoothness of the solution U of (5.8-9)) is not fulfilled. While we are solving (6.1) in D, we may concurrently monitor the coefficients of (6.4). Thus, J and $\frac{1}{J}\frac{\partial J}{\partial \xi}$ satisfy the ordinary differential equations,

(6.6) $$\frac{\partial J}{\partial \tau} = J\frac{\partial c}{\partial x}$$

and

(6.7) $$\frac{\partial}{\partial \tau}\left[\frac{1}{J}\frac{\partial J}{\partial \xi}\right] = J\frac{\partial^2 c}{\partial x^2}.$$

It follows from the construction of the characteristic coordinate system that the initial data for (6.6) is $J=1$ at $\tau=0$ for $\xi\geq 0$ and $J=1$ at $\tau=y_0^{-1}(\xi)$ for $\xi<0$. For $\xi\geq 0$ the initial data for (6.7) is clearly

$$\frac{1}{J}\frac{\partial J}{\partial \xi} = 0 \text{ at } \tau = 0.$$

The initial data for (6.7) for $\xi<0$ is a little more complicated, however. From the initial data for (6.6) it follows that the directional derivative of J is zero in the direction tangent to the curve $\tau=y_0(\xi)$. Also, it follows from (6.6) that along the curve $\xi=y_0(\tau)$ the directional derivative of J in the direction parallel to the τ-axis is equal to $\partial c/\partial x$. Thus, for $\xi<0$ the initial data for (6.7) is

$$\frac{1}{J}\frac{\partial J}{\partial \xi} = \frac{1}{c(0,\tau)}\frac{\partial c}{\partial x} \text{ at } \tau = y_0^{-1}(\xi).$$

There are, of course, many other ways to introduce a characteristic coordinate system, thus specifying initial data for (6.6) and (6.7). We have simply chosen one way which guarantees that the transformation to characteristic coordinates is C^1 in a neighborhood of the τ-axis. Let us also point out that (6.7) may be written in a form which clarifies the roles of the length scales L_c and L_{dc},

$$\frac{\partial}{\partial \tau}\left[\frac{\partial J}{\partial \xi}\right] = \frac{\partial J}{\partial \xi}\frac{\partial c}{\partial x} + J^2\frac{\partial^2 c}{\partial x^2}$$

We remark that when $\frac{\partial c}{\partial x}>0$, then $J\geq 1$ and that when $\frac{\partial c}{\partial x}<0$, then J can become arbitrarily small. Once J becomes as small as $O(\sqrt{\varepsilon})$, the diffusive effects are no longer negligible. This happens only for $\frac{\partial c}{\partial x}<0$, which describes a solution undergoing compression to form a steep gradient, indicating that the length scale for u is no longer equal to the length scale for γ. In the neighborhood of the steepening diffusion plays a dominant role. Thus by using J as a monitor, we determine dynamically where it is valid to neglect diffusion.

In conclusion, we have shown that asymptotic analysis of the convection-diffusion equation (5.1) leads to a decomposition of the domain into regions on which we should use different approximating equations and different numerical methods. The result is a numerical method which exploits concurrency inherent in the mathematics and identifies the synchronization required. The number of processors may be so large, though, that even more concurrency will have to be provided by the language.

Finally, let us remark that the method outlined above also applies to the nonlinear convection-diffusion equation obtained by allowing $\hat{c}$ to depend on $\hat{u}$ as well as on $\hat{x}$ and $\hat{t}$ in (5.2). In this case shock layers may develop in the interior of $\hat{D}$ (or, equivalently, in the interior of Δ, the image of D in the characteristic coordinate system). This phenomenon is quite analogous to the steepening of the gradient found in the linear problem when $\partial c / \partial x < 0$. The presence of shock layers requires the introduction of more subdomains. Note that as the solution progresses, we may monitor the need to introduce shock layers by concurrently computing the values of J and $\partial J / \partial \xi$ by solving the ordinary differential equation (6.6) and integrating (6.7).

Acknowledgments. The work of the first, second, and fourth authors was supported by the Applied Mathematical Sciences Research Program of the Office of Scientific Computing, Office of Energy Research, U. S. Department of Energy, by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48. The work of the third author was supported in part by the National Science Foundation under grant number DMS 8319783.

REFERENCES

[1] G. R. ANDREWS and F. B. SCHNEIDER, Concepts and Notations in Concurrent Programming, ACM Computing Surveys, 15, (1983), pp. 3-43.

[2] G. M. BAUDET, The Design and Analysis of Algorithms for Asynchronous Multiprocessors, Ph. D. Dissertation, Carnegie-Mellon University, Pittsburgh, 1978.

[3] R. C. Y. CHIN, G. W. HEDSTROM, and F. A. HOWES, A Survey of Analytical and Numerical Methods for Multiple-Scale Problems, UCRL-90971, Lawrence Livermore National Laboratory, Livermore, California, 1984.

[4] E. A. CODDINGTON and N. LEVINSON, Theory of Ordinary Differential Equations, McGraw Hill, New York, 1955.

[5] R. COURANT and D. HILBERT, Methods of Mathematical Physics, vol. II, Interscience, New York, 1962.

[6] W. ECKHAUS, Asymptotic Analysis of Singular Perturbations, North-Holland, Amsterdam, 1979.

[7] L. B. FELSEN and A. KAMEL, Hybrid Ray-Mode Formulation of *SH* Motion in a Two-Layer Half Space, Bull. Seismol. Soc. Amer., 71, (1981), pp. 1763-1781.

[8] A. R. GOURLAY, Hopscotch: A Fast Second-Order Partial Differential Equation Solver, J. Inst. Math. Appl., 6, (1970), pp. 375-390.

[9] D. HELLER, A Survey of Parallel Algorithms in Numerical Linear Algebra, SIAM Review, 20, (1978), pp. 740-777

[10] F. A. HOWES, Multi-Dimensional Reaction-Convection-Diffusion Equations, Proc. Conf. on Diff. Eqns., Dundee, Springer-Verlag, New York, 1984. (In press.)

[11] J. KEVORKIAN and J. D. COLE, Perturbation Methods in Applied Mathematics, Springer-Verlag, New York, 1981.

[12] U. KIRCHGRABER, Dynamical System Methods in Numerical Analysis. Part I: An ODE Solver Based on the Method of Averaging, Research Rept. 83-02, Seminar für Angewandte Mathematik, Eidgenössische Technische Hochschule, CH-8092 Zürich, 1983.

[13] D. J. KUCK, R. H. KUHN, D. A. PADUA, B. LEASURE, and M. WOLFE, Dependence Graphs and Compiler Optimization, Proc. 8th ACM Symposium on Principle of Programming Languages, (1981), pp. 207-218.

[14] J. R. MCGRAW, The VAL Language: Description and Analysis, ACM Transactions on Programming Languages and Systems, 4, (1982), pp. 44-82.

[15] A. H. NAYFEH, Perturbation Methods, Wiley-Interscience, New York, 1973.

[16] N. S. OSTLUND, P. G. HIBBARD, and R. A. WHITESIDE, A Case Study in the Application of a Tightly Coupled Multiprocessor to Scientific Computations, pp. 315-364 in Parallel Computations, ed. G. Rodrigue, Academic Press, New York, 1982.

[17] L. R. PETZOLD, An Efficient Numerical Method for Highly Oscillatory Ordinary Differential Equations, SIAM J. Numer. Anal., 18, (1981), pp. 455-479.

[18] A. H. SAMEH, Numerical Parallel Algorithms - A Survey, pp. 207-228 in High Speed Computer and Algorithm Organization, ed. D. J. Kuck, D. H. Lawrie, and A. H. Sameh, Academic Press, New York, 1977.